Servicio de comunicación causal bidireccional sin contención

Isabel Muñoz, Sergio Arévalo

Escuela Universitaria de Infórmatica
Universidad Politécnica de Madrid
Carretera de Valencia Km 7
28031 Madrid
{imunoz,sarevalo}@eui.upm.es

Abstract. Las arquitecturas jerárquicas de comunicación causal se presentan como una alternativa habitual para reducir el elevado tamaño de la información de control causal a enviar en cada mensaje, cuando la comunicación se realiza entre un subconjunto de procesos que pertenecen a un grupo muy numeroso. Sin embargo, en estas arquitecturas, los nodos intermedios de la jerarquía padecen un efecto indeseable denominado efecto convoy. Estos nodos intermedios tienden a generar ráfagas de envíos que sobrecargan tanto a los nodos de los niveles inferiores de la jerarquía como a la red, provocando pérdidas de mensajes y periodos entre ráfagas de infrautilización de la red. Este artículo presenta un servicio causal bidireccional sin contención que, aplicado a los nodos intermedios de la jerarquía, soluciona el efecto convoy. Este servicio causal sin contención entrega a la capa de aplicación y envía al sistema un mensaje sin esperar la entrega o el envío previo de mensajes que constituyen la historia causal del primero, por lo que evita las ráfagas de entrega y de envío de mensajes. La entrega de un mensaje va acompañada de un identificador causal, que es un número natural que indica el número de orden de ese mensaje en la secuencia causal total. El envío de un mensaje supone construir un vector causal válido a partir de un identificador causal, que permita ordenar dicho mensaje en orden causal en el proceso receptor.

1 Introducción

Un sistema distribuido está formado por un conjunto de procesos cooperantes ubicados en nodos separados geográficamente, que intercambian información mediante el paso de mensajes. Cuando los canales de comunicación son asíncronos, los procesos pueden ser incapaces de determinar el orden temporal en el que se producen los sucesos, a la vista de los mensajes que van recibiendo. Una forma de reducir la asincronía del canal es la inclusión de una capa, entre el canal y el proceso, que asegure a éste un determinado orden de entrega de los mensajes. En concreto, una capa de comunicación causal [4] garantiza la entrega de mensajes en el orden causal en el que se han enviado. Sin embargo, para poder determinar el orden de entrega de un mensaje, es necesario incluir información de control

en éste. El tamaño de la información de control causal dependerá de si la comunicación es "uno a todos" (multidifusión o multicast), "uno a varios" o "uno a uno". Así, cuando el mensaje se difunde a todos los miembros de un grupo, el coste de la información de control causal es $\theta(n)$ [11], siendo n el tamaño del grupo. Sin embargo, si el mensaje se envía a un subconjunto del grupo total, el coste de la información de control causal a enviar es $\theta(n^2)$ [10]. Es evidente que el inconveniente de la comunicación causal entre un subconjunto elevado de procesos es el coste de la transmisión de la información de control con respecto a la escalabilidad del sistema. Entre las diferentes soluciones propuestas para reducir el tamaño de la información de control destacan los modelos basados en arquitecturas jerárquicas, como la arquitectura en forma de margarita propuesta por Aldi y Nagi [1] y Baldoni [2]. En esta arquitectura, los procesos de usuarios se reparten en grupos disjuntos conectados por un grupo central de procesos especiales, denominados servidores causales, que se encargan de difundir los mensajes generados en un grupo al resto de los grupos de procesos, utilizando multidifusión. Sin embargo, aunque la organización jerárquica y el uso de la multidifusión reducen el coste causal a un coste lineal $\theta(m)$ siendo m la cardinalidad del grupo más grande, introduce un efecto indeseable conocido como efecto convoy. El efecto convoy se produce cuando a un proceso receptor se le entrega una ráfaga de mensajes que produce una sobrecarga de procesamiento, aunque los procesos emisores hayan realizado el envío a una tasa constante. Aunque Gupta [5] indica que este efecto se puede producir en cualquier tipo de arquitectura como consecuencia de situaciones transitorias debidas al control de flujo o debidas al uso de protocolos de comunicación fiable, Kalantar [7] indica que este efecto se acentúa sobremanera en arquitecturas jerárquicas con orden de entrega causal o total, cuando los nodos se comunican mediante multidifusión. La llegada a un nodo intermedio de la jerarquía de un mensaje fuera de orden supone que la entrega de éste debe retrasarse hasta la llegada de los mensajes que le preceden causalmente. Esto puede dar lugar a la entrega de la secuencia completa en forma de ráfaga. Así, es posible que a su vez el receptor envíe una nueva ráfaga, incluso más larga, al siguiente nivel de la jerarquía, amplificándose el efecto a medida que se desciende en ésta y, por tanto, aumentando la probabilidad de control de flujo en los receptores, de pérdidas de mensajes y de la necesidad de retransmisiones. Además, el rendimiento del canal disminuye ya que el canal puede llegar a estar ocioso en los periodos entre ráfagas y sin embargo durante éstas el canal no tenga capacidad suficiente para transmitirlas. La magnitud del efecto convoy puede ser tan perniciosa de cara al rendimiento del sistema, que grandes compañías como eBay o Google dedican grandes esfuerzos en intentar reducir dicho efecto [3].

En este artículo proponemos un protocolo de comunicación bidireccional causal sin contención. Es bidireccional ya que evita la contención tanto en la entrega como en el envío de mensajes, solucionando así el efecto convoy que se produce en arquitecturas jerárquicas con orden de entrega causal. En nuestro modelo causal sin contención, el protocolo de comunicación causal entregará o enviará un mensaje inmediatamente, aunque esté desordenado causalmente. Para preser-

var la semántica causal en cuanto a la entrega, cada mensaje irá acompañado de un identificador que indica el orden que ocupa dicho mensaje en la secuencia causal. El algoritmo para el cálculo de dicho identificador es el propuesto por Muñoz y Arévalo [9]. Por otra parte, para permitir el envío de un mensaje desde la capa de aplicación hacia el resto del sistema sin haber mandando previamente los mensajes que le preceden causalmente, en este artículo se propone un algoritmo que partiendo del identificador causal del mensaje construye un vector causal válido con el que etiquetar el mensaje. Con respecto a los protocolos de envío causal con contención en arquitecturas jerárquicas, este protocolo de envío causal sin contención presenta las siguientes ventajas: 1) requiere sólo el almacenamiento condensado de la información causal y de los identificadores de los mensajes recibidos, frente al almacenamiento de mensajes fuera de secuencia causal necesarios en los protocolos de entrega causal con contención 2) evita el efecto convoy, ya que al no haber contención en la entrega ni en le envío del mensaje se retransmite sin demora al nodo siguiente, 3) reduce la probabilidad de pérdida de mensajes ya que al no generar ráfagas de envío no se producen sobrecargas de búferes en los nodos receptores y 4) aprovecha de manera más uniforme la capacidad del canal, ya que no tiende a generar tiempos muertos entre envíos. El resto del artículo se estructura como sigue. La sección 2 describe el modelo de sistema y los principales conceptos relacionados con la entrega causal, en la sección 3 se presenta el protocolo de envío causal sin contención que, junto con la entrega de mensajes sin contención, soluciona el efecto convoy. La sección 4 muestra las conclusiones y las futuras líneas de trabajo.

2 Modelo de sistema y definiciones

2.1 Sistema distribuido asíncrono

El sistema distribuido considerado está compuesto por un conjunto finito P de n procesos, que no comparten memoria ni reloj, conectados a través de una red fiable de comunicación sobre la que intercambian un conjunto de mensajes M mediante multidifusión o multicast. El tiempo de transmisión de un mensaje es finito pero no acotado. Cada par ordenado de procesos (p_i, p_j) está conectado por un canal fiable c_{ij}, a través del cual p_i puede enviar mensajes a p_j. El canal es fiable, por lo que se considera que no se recibirán mensajes duplicados ni se perderán mensajes, pero si puede ocurrir que los mensajes lleguen desordenados. Se supone que tanto el canal de comunicación como los procesos no van a sufrir ningún tipo de fallo.

2.2 Relación de precedencia causal "Sucedió antes"

Lamport definió la relación causal "Sucedió antes" en [8]. Dicha relación determina que, dados dos sucesos, a y b, si "a sucedió antes que b", representándose $a \rightarrow b$, entonces todos los procesos coinciden en que primero se produjo el suceso a y posteriormente el b. Esta relación es cierta si y solo si:

1. Los sucesos a y b se producen en el mismo proceso y a se produce antes que b.
2. Si a es un suceso de envío de un mensaje m y b es el suceso de recepción de dicho mensaje.
3. Existe un suceso c, tal que $a \rightarrow b$ y $b \rightarrow c$. La relación "sucedió antes" es transitiva.

Si dos sucesos, x e y, se producen respectivamente en dos procesos que no intercambian mensajes ni directa ni indirectamente, entonces, no es cierto que $x \rightarrow y$ ni que $y \rightarrow x$. En este caso, se dice que tales sucesos son concurrentes, representándose por $x||y$, indicando que no se sabe nada sobre cuál de ellos sucedió primero.

2.3 Orden de entrega causal

Hadzilacos y Toueg [6] consideran que se produce entrega causal de dos mensajes m y m', representándose $m \rightarrow m'$, cuando la difusión del mensaje m precede causalmente a la difusión del mensaje m' y ningún proceso entrega m' a menos que previamente haya entregado m. Los mensajes que no cumplan $m \rightarrow m'$ o $m' \rightarrow m$, son mensajes concurrentes entre sí y han podido entregarse en cualquier orden.

2.4 Identificador causal de un mensaje

Sea $[I]$ el intervalo cerrado de los números naturales $[1, |M|]$ donde $|M|$ es la cardinalidad del conjunto de los mensajes enviados en el sistema M en una ejecución σ. La función de identificador causal (cId) definida en [9] es tal que asigna un número natural, denominado identificador causal, a cada mensaje recibido por un determinado proceso. El identificador causal será un número natural comprendido en el intervalo $[1, |M|]$.

$$cId : PxM \rightarrow I$$

La función cId_p es la restricción de la función cId en el conjunto de mensajes recibidos por un proceso $p \in P$. Esta función es una función uno a uno.

$$cId_p : M \rightarrow I$$

La función cId_p presenta las siguientes propiedades:

Property 1 (Unicidad). Para un proceso p y para cualquier par de mensajes m y m' es cierto que $cId_p(m) \neq cId_p(m')$.

Property 2 (Correspondencia causal). Si un mensaje m precede causalmente a otro mensaje m' entonces el identficador causal de m será menor que el identificador causal de m' para todo proceso p.

$$m \rightarrow m' \Rightarrow cId_p(m) < cId_p(m')$$

Property 3. (No agujeros causales) Los identificadores causales de los mensajes entregados a un proceso p en una ejecución del sistema deben de ser mayores que 0 y menores o iguales que el número de mensajes M enviados en la ejecución del sistema.

$$1 \leqslant cId_p(m) \leqslant |M|$$

2.5 Servicio de comunicación causal con contención

El servicio de comunicación causal con contención mostrado a continuación se basa en el servicio clásico de entrega causal. Dicho servicio entrega los mensajes en orden causal, almacenado los mensajes se reciban desordenados hasta que puede entregar un secuencia ordenada de mensajes. Se considera que cada proceso en el sistema está compuesto por un módulo de entrega causal que interactúa con la aplicación y el servicio de multicast subyacente.
Cada acción del servicio se parametriza por un único proceso $p \in P$ en el que la acción se produce. Para describir dicho servicio se han utilizado los siguientes tipos:

P es el conjunto de procesos del sistema
M es el conjunto de mensajes enviados por los procesos
V es el conjunto de vectores causales de dimensión P

La firma externa del servicio se compone de las siguientes acciones:

- input **send** $(p, m), p \in P, m \in M$
  La capa de aplicación del proceso p envía un mensaje m.
- output **deliver** $(p, m), p \in P, m \in M$
  El proceso p entrega en orden causal el mensaje m.
- output **net_send** $(p, < m, v_m >), p \in P, m \in M, v_m \in V$
  El proceso p difunde un mensaje m con una marca de tiempo v_m a través de la capa de comunicación subyacente.
- input **net_receive** $(p, < m, v_m >), p \in P, m \in M, v_m \in V$
  el proceso p recibe un mensaje m junto con su marca de tiempo v_m de la red subyacente.

El módulo de aplicación puede generar sucesos de tipo *send* al módulo causal, mientras que admite de éste sucesos de tipo *deliver*. Por su parte, el módulo causal generará sucesos de tipo *net_send* hacia el canal y recibirá de éste sucesos de tipo *net_receive*. La entrega de un mensaje en orden causal se produce como consecuencia de un suceso del tipo *deliver*, mientras que la multidifusión causal (causal multicast) de un mensaje se produce derivado de un suceso de tipo *net_send*. La recepción de un mensaje se produce como consecuencia de un suceso *net_receive* y finalmente, la multidifusión de un mensaje se produce como consecuencia de un suceso *send*. Los sucesos *deliver* y *net_send* solo se pueden producir como consecuencia de sucesos previos *net_receive* y *send* respectivamente.

Una ejecución del sistema distribuido σ es una colección de sucesos *send* y *deliver* ordenados según la relación de orden parcial "Sucedió antes". Sea M el conjunto de mensajes entregados en σ, dicha ejecución respetará el orden de entrega causal si para cualquier mensaje $m \in M$ se cumple la propiedad de orden de entrega causal descrita anteriormente.

2.6 Algoritmo de entrega causal con contención

La causalidad de la multidifusión puede capturarse mediante el uso de relojes lógicos vectoriales. El sistema de relojes lógicos vectoriales asocia marcas de tiempo a los sucesos de envío de mensajes, de tal forma que, comparando las marcas de tiempo de dos sucesos de envío, se puede saber si están causalmente relacionados [9]. Cada proceso p mantendrá en su módulo causal un reloj lógico vectorial $v_p \in V$. El vector de enteros $v_p[1, ..., n]$ está indexado por identificador de proceso e inicializado a $[0, ..., 0]$. El módulo causal actualizará dicho vector según las reglas descritas para el algoritmo CBCAST [11]:

1. Para cada suceso $send(p, m)$, el proceso p:

   (a) incrementa su contador en el vector v_p: $v_p[p] = v_p[p] + 1$
   (b) genera un suceso $net\_send(p, m, v_p)$

2. Cuando un proceso q recibe un mensaje m, tras un suceso $net\_receive(q, m, v_m)$ difundido por $p \neq q$, no generará el suceso $deliver(m, v)$ de m hasta que se cumplan las dos condiciones siguientes:

   (a) $v_m[p] = v_q[p] + 1$
   (b) $\forall k, 1 \leq k \neq p \leq n : v_m[k] \leq v_q[k]$

   La condición 2.(a) indica que los mensajes del emisor p se deben entregar en orden FIFO en q, mientras que la condición 2.(b) indica que para poder entregar m, necesariamente se han tenido que entregar en q los mensajes que preceden causalmente a m.

3. El suceso de entrega $deliver(q, m)$ modifica el vector causal del receptor q de acuerdo a la marca de tiempo v_m de m, de la siguiente manera: $\forall k, 1...n : v_q[k] = max(v_q[k], v_m[k])$

Al final de una ejecución del sistema σ, a cada proceso p se le habrá entregado una secuencia causal sc_p formada por el conjunto M de mensajes. Sin embargo, para dos procesos cualesquiera p y q, el orden de los mensajes en las secuencias sc_p y sc_q, puede no coincidir ya que por la asincronía de la red, los mensajes concurrentes pueden llegar y, por tanto, entregarse en cualquier orden. Obsérvese que en el paso 2 del algoritmo se produce contención causal en la entrega. Es decir, cuando se recibe un mensaje fuera de secuencia, no se entrega a la capa de aplicación hasta que no se ha recibido toda la historia causal de dicho mensaje.

3 Módulo de comunicación causal sin contención

El servicio de comunicación causal sin contención entrega cada mensaje m que recibe del canal de comunicación de forma inmediata, junto un número natural que será su identificador causal obtenido de la función $cId_p(m)$. Dicha función cumple las propiedades descritas en el apartado 2.4. Puesto que los mensajes pueden entregarse desordenados, el identificador causal permite a la aplicación ordenar causalmente los mensajes, ordenándolos en orden creciente de identificador. Por otra parte, el servicio permite que la aplicación envíe cualquier mensaje etiquetado con un identificador causal. A la hora de enviar dicho mensaje al sistema, el servicio transforma dicho identificador en una marca de tiempo en forma de vector.

Para describir este servicio se han utilizado los siguientes tipos:

P es el conjunto de procesos del sistema
M es el conjunto de mensajes enviados por el sistema
V es el conjunto de vectores causales de dimensión P
I es el intervalo cerrado de números naturales $[1, |M|]$ donde $|M|$ es la cardinalidad del conjunto M

El servicio de comunicación causal sin contención presenta la siguiente interfaz para interactuar con la aplicación y la red subyacente.

- input **cSend** $(p, < m, cId_p(m) >), p \in P, m \in M, cId_p(m) \in I$
  La capa de aplicación en p envía un mensaje m junto con un identificador causal $cId_p(m)$.
- output **cDeliver** $(p, < m, cId_p(m) >), p \in P, m \in M, cId_p(m) \in I$
  El proceso p entrega el mensaje m y su identificador causal $cId_p(m)$.
- output **net_send** $(p, < m, v_m >), p \in P, m \in M, v_m \in V$
  El proceso p envía un mensaje m y su marca de tiempo v_m a través de la red subyacente.
- input **net_receive** $(p, < m, v_m >), p \in P, m \in M, v_m \in V$
  El proceso p recibe un mensaje m y su marca de tiempo v a través de la red.

Obsérvese que cuando se produce la recepción de un mensaje m tras un suceso $net\_receive(p, < m, v_m >)$, el servicio debe transformar la marca de tiempo vectorial v_m en un identificador causal $cId_p(m)$, para poder generar un suceso de entrega $cDeliver(p, < m, cId_p(m) >)$. De forma simétrica, cuando se produce el suceso $cSend(p, < m, cId_p(m) >$, el servicio debe transformar el identificador causal $cId_p(m)$ asociado con m en una marca de tiempo vectorial v_m para poder generar un suceso de envío por la red $net\_send(p, < m, v_m >)$.
La propiedad que debe cumplir dicho servicio es la de preservar la causalidad. Es decir, si un mensaje m' precede causalmente a otro m, entonces la marca de tiempo del primero será menor que la del segundo:

$$m' \rightarrow m \Rightarrow v_{m'} < v_m$$

Para el cálculo del identificador causal $cId_p(m)$ a partir de la marca de tiempo v_m se propone aplicar el algoritmo propuesto en [9], donde se verifica que:

$$m' \rightarrow m \Rightarrow cId_p(m') < cId_p(m)$$

La transformación del identificador causal $cId_p(m)$ de m en una marca de tiempo v_m debe preservar la información causal. Por tanto debe verificarse que:

$$cId_p(m') < cId_p(m) \Rightarrow v_{m'} < v_m$$

4 Algoritmo de reenvío causal rápido

El algoritmo que se va a describir en este apartado permite la transformación de un identificador causal $cId_p(m)$ de un mensaje m en una marca de tiempo vectorial v_m. Es importante destacar que es necesario que el identificador causal sea válido, esto es, que cumpla la propiedades de unicidad, correspondencia causal y de no generación de agujeros causales descritos en el apartado 2.4. para poder preservar la causalidad del sistema.
La idea fundamental sobre la que se sustenta el algoritmo es la siguiente: Sea m un mensaje cuyo identificador causal es $cId_p(m)$ para el proceso p. Si p envía m hacía el sistema con una marca de tiempo v_m donde $v_m(p) = cId_p(m)$, todos aquellos mensajes m' que se envíen posteriormente y que precedan causalmente a m, deberán enviarse con una marca de tiempo $v_{m'}$ menor que v_m. Es decir:

$$cId_p(m') < cId_p(m) \Rightarrow v_{m'} < v_m$$

En la parte declarativa del algoritmo, tras la definición de los conjuntos de procesos, mensajes y relojes vectoriales, se observa la definición de la lista de mensajes fuera de secuencia LSent que se describirá más adelante. Después se definen los tipos de mensajes a utilizar. El primero, $appMsg$ es el tipo de mensaje que envía la capa de aplicación al servicio de entrega causal ($A \rightarrow S$). Dicho mensaje estará formado por los datos msg del mensaje junto con un número natural cId que será su identificador causal. El segundo mensaje $fcMsg$ es el que entregaría la capa de servicio a la aplicación cuando reciba un mensaje($S \rightarrow A$). Finalmente, el servicio causal enviará mensajes $netMsg$ por la red ($S \rightarrow N$) o bien recibirá mensajes de la red ($N \rightarrow S$) constituidos por los datos msg y etiquetados con una marca de tiempo vectorial v.

En cuanto a las variables usadas, se destaca la lista S de tipo $LSent$. S es una lista de nodos ordenados en orden creciente de identificador causal. Cada nodo s representa un rango de identificadores causales de mensajes que no han llegado todavía desde la capa de aplicación, pero a los que ya se les ha asignado un marca de tiempo vectorial. Dicha marca vectorial corresponde a un mensaje que les poscede causalmente y que ya ha sido enviado. Esta marca de tiempo se usará para etiquetar dichos mensajes cuando se envíen por la red. El nodo cuenta con tres campos, como indica el tipo $Sent$. El primer campo, cId representa el mayor

```
Tipos:
   Type P SetOf(procesos)
   Type M SetOf(mensajes)
   Type V SetOf(vectores causales)
   Type CId SetOf (identificadores causales)
   Type Sent <CId cId, int n,V v>
   Type LSent sequenceOf(Sent)

Tipos de mensajes:
   A →S  appMsg = <M msg, CId cId>
   S→ A  fcMsg = <M msg, CId cId>
   S →N, N →S netMsg = <M msg, V v>

Variables y estructuras de datos
   P  p  // Identificador del proceso p
   LSent S  // Lista de mensajes enviados fuera de secuencia
   V vClock  // Reloj vectorial de p
   Sent  s  // Elemento de la lista LSent
   V v_m   // Marca de tiempo del mensaje m a enviar

Funciones externas:
   sucesorCausal [LSent, CId →Sent]
   predecesor [LSent, Sent →Sent]
   esPrimerNodo[Lsent, Sent →boolean]
   ultimoNodo[LSent →Sent]
   crearNodo[LSent, CId, V, int →Sent]
   eliminarNodo[LSent, Sent →LSent]
```

Fig. 1. Tipos y variables para el envío causal sin contención

```
1  Upon reception appMsg m in p:
2
3     if vacía (S) and (vClock[p] +1 = m.cId) then // m está en secuencia
4        vClock[p] ← vClock[p]+1 // se actualiza el reloj de p
5        net_send < m.msg, vClock >
6
7     else // m no está en secuencia
8        s ← sucesorCausal(S, m.cId)
9        if (s = ⊥ ) then  // m no tiene sucesor causal en S
10          s = ultimoNodo(S)
11          if (s = ⊥ ) then  //S está vacía
12             s = crearNodo(S, m.cId, vClock, m.cId − vClock[p] )
13          else     // S no está vacía
14             if (s.v = vClock) then // no ha cambiado el reloj. Uso s
15                s.n = s.n + m.cId − s.cId
16                s.cId = m.cId
17             else  // ha cambiado el reloj. Se crea un nodo s en S
18                s = crearNodo(S, m.cId, m.cId − ultimo(S).cId, vClock)
19             endif
20          endif
21       endif
22
23       v_m = s.v     //se construye y envía el mensaje
24       v_m[p] = m.cId
25       net_send< m.msg, v_m >
26
27       s.n = s.n −1     //un mensaje menos a enviar
28       if (s.n = 0) then // enviados todos los mensajes del nodo s
29          if esPrimerNodo(S, s)) then // s en secuencia con el reloj
30             vClock[p] ← s.cId // se actualiza el reloj al cId de s
31          else // s es un nodo intermedio. Se une al anterior
32             pred ← predecesor(S, s)
33             pred.cId ← s.cId
34          endif
35          eliminarNodo (S, s)
36       endif
37
38    endif
```

Fig. 2. Protocolo para el envío causal sin contención

identificador causal de los mensajes asociados a dicho nodo. El segundo campo n es el número de mensajes pendientes de recepción que deben enviarse con la marca de tiempo del nodo s. Un mensaje m se enviará con dicha marca de tiempo si su identificador causal $m.cId$ cumple: $s.cId - s.n < m.cId < s.cId$. El tercer campo contiene la marca de tiempo vectorial v con el que habrá que etiquetar dichos mensajes. En cuanto a la declaración de funciones, la más destacable es la función *sucesorCausal* que tomando como entrada la lista S y un identificador causal $m.cId$ de un mensaje m recorre la lista buscando el menor nodo n cuyo identificador causal $s.cId$ sea mayor que $m.cId$. La función predecesor devuelve el nodo, si existe, que precede a un nodo en una secuencia S. El resto de funciones declaradas se utilizan para manipular la lista S.

Observando el seudocódigo propuesto, cuando la capa de aplicación del proceso p produce un suceso del tipo *cSend* el servicio de comunicación causal rápida recibe un mensaje $m =< msg, cId >$, siendo $m.msg$ los datos del mensaje y $m.cId$ el identificador causal del mensaje para p (linea 1). Para que el servicio pueda enviar dicho mensaje a través de la red, necesita calcular la marca de tiempo vectorial del mensaje a partir del reloj vectorial del proceso p y del valor del identificador causal del mensaje para p. No se van a utilizar subíndices para el proceso p para no oscurecer el código. El caso más simple es que el mensaje esté en secuencia causal. Esto es, el identificador $m.cId$ de m es el siguiente al último enviado en secuencia según el reloj vectorial de p. Por tanto, se incrementa en uno el reloj en la posición de p y se envía por la red, etiquetando el mensaje con el valor del reloj (líneas 3 a 5). Si el mensaje no está en secuencia, esto indica que se va a enviar un mensaje tal que algún o algunos de los mensajes que le preceden causalmente no se han enviado todavía. Si es el primer mensaje fuera de secuencia, entonces la lista S de mensajes enviados fuera de secuencia está vacía, por que se crea un nuevo nodo s. En s el mayor identificador causal será el del mensaje $m.cId$, la marca de tiempo o timestamp será el valor actual del reloj $vClock$ en p y el número de mensajes pendientes de envío con dicha marca de tiempo serán tantos como la diferencia entre el identificador causal del mensaje $m.cId$ y el último mensaje enviado en secuencia por p en $vClock[p]$ (Lineas de 11 a 12). Por otra parte, si la lista de mensajes fuera de secuencia S no estuviera vacía, podrían plantearse dos alternativas, dependiendo de si m cuenta con sucesor causal en la secuencia S (Línea 8). En primer lugar, supongamos que m cuenta con sucesor causal s en S. El sucesor causal será el menor elemento de s tal que $m.cId < s.cId$. Entonces, m deberá mandarse con la marca de tiempo que indique s, actualizando la posición de p con el valor del identificador de m (Líneas 23-25).
En segundo lugar, supóngase que m no cuenta con sucesor causal en S porque m es el mensaje con mayor identificador causal fuera de secuencia. Es decir, el último elemento s en S cumple que $s.cId < m.cId$. En esta situación, pueden darse dos casos. Supóngase que la marca de tiempo de s tiene el mismo valor que el reloj de p, en este caso se puede modificar este nodo para que el mayor identificador causal asociado incluya hasta el identificador de m. Por tanto, habrá que incrementar el contador de mensajes fuera de secuencia con aquellos que no

han llegado y que posceden a s y preceden a m (Líneas 14-16). Por otra parte, si la marca de tiempo de s es menor que el valor actual de reloj de p es necesario crear un nuevo nodo (Línea 18) con la nueva marca de tiempo.
Tanto si m tiene o no sucesor causal, se termina enviando el mensaje por la red (Líneas 23-26). Finalmente, queda explicar qué ocurre cuando en un nodo s se han enviado ya todos los mensajes fuera de secuencia. Esta condición se cumple cuando $n.s = 0$. Cuando el nodo s es el primero de la lista y ya ha enviado con la marca de tiempo asociada $s.v$ los n mensajes pendientes,entonces se actualiza el reloj p, indicando que p hasta el momento ha enviado en secuencia hasta $s.cId$ mensajes eliminándose el nodo s de S (Líneas 29-30). En otro caso, simplemente se incrementa el identificador causal de nodo predecesor de s para después eliminar el nodo s.

4.1 Ejemplo de ejecución del algoritmo

Sea $G = (p, q, r)$ un grupo de procesos. Sea p un proceso con un módulo de comunicación causal rápida. Supóngase que en una ejecución del sistema, se produce en el proceso p una secuencia de sucesos de tipo $cSend(m, cId_p(m))$ generados por la capa de aplicación de p y $net\_receive(m, v_m)$ generado por la capa de comunicación de p. En estos tipos de eventos considérese que m es el mensaje, $cId_p(m)$ el identificador causal asociado a m en p y v_m la marca de tiempo vectorial del mensaje que se recibe por la red.
Considerando la secuencia siguiente:
$cSend(m_3, 3), cSend(m_4, 4), cSend(m_1, 1), net\_receive(m_r, [0, 0, 1]), cSend(m_5, 5)$
$cSend(m_2, 2)$
Se verá qué sucesos produce el módulo de comunicación rápida como resultado del procesamiento de los eventos anteriores. Inicialmente el proceso p tiene su reloj vectorial $vClock = [0, 0, 0]$, puesto que p no ha enviado ningún mensaje y no ha recibido ningún mensaje procedente ni de q ni de r respectivamente. La secuencia S de mensajes enviados fuera de secuencia está vacía.

- Se produce el suceso $cSend(m_3, 3)$. El identificador causal 3 no está en secuencia con el último mensaje enviado en orden causal por p. Es decir, $vClock[p] + 1 <> 3$, por tanto se crea un nuevo nodo s_1 y se inserta en S. $S = (s_1 =< 3, [0, 0, 0], 3 >)$, donde 3 es el identificador causal mayor del nodo, el vector es la marca de tiempo de referencia y 3 es el número de mensajes pendientes de envío con esta marca de tiempo (Línea 12 del algoritmo). Después se construye la marca de tiempo del mensaje m_3, copiando el valor del vector del nodo y modificando el valor del emisor p con el valor del identificador del mensaje. Así, se genera un suceso de envío $net\_send < m_3, [3, 0, 0] >$ (Líneas 23-25). Se decrementa en 1 los mensajes pendientes por enviar en dicho nodo, quedando $S = (s_1 =< 3, [0, 0, 0], 2 >)$ puesto que 2 es el número de mensajes pendientes, correspondientes a los mensajes con identificadores causales 1 y 2.

- Se produce el suceso $cSend(m_4, 4)$. El mensaje no está en secuencia con el último mensaje enviado por p por lo que se busca en S si existe un nodo sucesor cuyo identificador causal sea mayor que 4. No existe. Se pasa a comprobar si puede anexar dicho identificador con el nodo ya existente (L-14). Esto es posible ya que el reloj vectorial de p sigue siendo $[0, 0, 0]$ y coincide con el vector del nodo s_1. Así, se modifica el nodo s_1 resultando $S = (s_1 =< 4, [0, 0, 0], 3 >)$. Se envía el mensaje como $net\_send < m_4, [4, 0, 0] >$ y se decrementa el número de mensajes pendientes de envío, resultando $S = (s_1 =< 4, [0, 0, 0], 2 >)$

- Se produce el suceso $cSend(m_1, 1)$. El mensaje está en secuencia con el reloj vectorial de p, pero la secuencia S no está vacía. Se busca el sucesor causal resultado ser de nuevo s_1 (L-8). Se pasa a construir el mensaje y se genera el suceso $net\_send < m_1, [1, 0, 0] >$ y se decrementa el número de mensajes pendientes de envío en s_3 resultando $S = (s_1 =< 4, [0, 0, 0], 1 >)$.

- Se produce el suceso $net\_receive(m_r, [0, 0, 1])$ procedente del proceso r. Puesto que la recepción del mensaje de r está en secuencia con el reloj vectorial de p, se actualiza el reloj de p como $vClock[r] = vClock[r] + 1$, resultando $vClock = [0, 0, 1]$. El módulo calculará el identificador causal asociado al vector [0,0,1] según el algoritmo indicado en [9] dando como resultado que $Idc(m_r) = 1$. Finalmente generaría el suceso $cDeliver(m_r, 1)$ a la aplicación.

- Se produce el suceso $cSend(m_5, 5)$. Se busca un sucesor causal en S. No existe y además no se puede anexar al nodo existente ya que el vector del nodo s_1 no coincide ya con el reloj vectorial de p. Por tanto se crea un nuevo nodo (L-18) resultado $S = (s_1 =< 4, [0, 0, 0], 1 >, s_2 =< 5, [0, 0, 1], 1 >)$. Se envía el mensaje mediante $net\_send < m_5, [5, 0, 1] >$, se decrementa el número de mensajes pendientes en el nodo s_2, resultando que $n = 0$. (L-28). Por tanto, se compacta la lista S asignando el 5 como identificador de s_1 y eliminando el nodo s_2. Así $S = (s_1 =< 5, [0, 0, 0], 1 >)$.

- Se produce el suceso $cSend(m_2, 2)$. El nodo sucesor es s_1, se construye y envía el mensaje y se genera el suceso $net\_send < m_2, [2, 0, 0] >$. Se decrementa los mensajes pendientes en s_1. Ahora ya no queda ningún mensaje por enviar y el nodo es el primero de la lista por lo que se actualiza el reloj vectorial de p con el mayor identificador causal de s_1. Así $vClock[p] = 5$ y por tanto $vClock = [5, 0, 1]$. Se elimina el nodo s_1 y la lista S se queda vacía.

  Por tanto, un proceso receptor podrá ordenar en orden causal los mensajes recibidos aplicando las reglas de entrega causal con contención:
  $< m_1, [1, 0, 0] >, < m_2, [2, 0, 0] >, < m_3, [3, 0, 0] >, < m_4, [4, 0, 0] >, < m_5, [5, 0, 1] >$

5 Conclusiones

En este artículo se ha descrito un servicio de entrega y envío causal sin contención que soluciona el problema del efecto convoy que presentan las arquitecturas jerárquicas de comunicación causal. Dicho módulo se aplica en los nodos intermedios de dicha arquitectura que retransmiten en ambas direcciones los mensajes que les llegan procedentes de los grupos en los que actúan como pasarelas. El uso de este módulo bidireccional sin contención es transparente en los nodos hoja ya que no impide que en éstos se sigan usando servicios de entrega tradicionales de comunicación causal con contención. El servicio expuesto, realiza la entrega causal sin contención de cada mensaje tras su recepción, sin esperar a entregar previamente los mensajes de la historia causal de dicho mensaje. La entrega de cada mensaje va acompañada de un identificador causal, que es un número natural que indica el número de orden de ese mensaje en la secuencia causal completa. Por otro lado, se propone un algoritmo para el envío al sistema de mensajes procedentes de la capa de aplicación que no vengan en orden causal, pero que vengan acompañados de un identificador causal válido. El algoritmo calcula una marca de tiempo válida, en forma de vector causal, a partir del identificador causal del mensaje y del valor del reloj lógico del proceso. Con dicha marca de tiempo vectorial se etiqueta al mensaje que va a viajar por la red. Internamente el protocolo solo necesita mantener una lista de subsecuencias o rangos de identificadores asignados y un vector causal por cada rango. El protocolo, insertado en un módulo de entrega causa bidireccional sin contención, ya se ha probado y nos encontramos en la fase de evaluar su rendimiento, que esperamos lógicamente que mejore sustancialmente cualquier servicio causal con contención tanto en la entrega como en el envío. Por otra parte, se pretende modificar el algoritmo para que realice filtrado de mensajes cuando se utilice multicast con autoenvío así como conseguir aplicar dicho algoritmo a arquitecturas que soporten más de un proceso pasarela entre islas para soportar tolerancia a fallos.

References

[1] Noha A., Magdy N.: Maintaining Causal Order in Large Scale Distributed Systems Using a Logical Hierarchy. Proc. IASTED Int. Conf. on Applied Informatics. (1995) 214–219

[2] Baldoni, R., Friedman, R. and Van Renesse, R.: The Hierarchical Daisy Architecture for Causal Delivery. 17th IEEE International Conference on Distributed Computing Systems IEEE Computer Society, Washington, DC, USA (1997) 570

[3] Birman, K., Chockler, G., and van Renesse, R.: Toward a Cloud Computing Research Agenda. J. SIGACT News 40, 2 (2009) 68-80

[4] Birman, K. P. and Joseph, T. A. Exploiting Virtual Synchrony in Distributed Systems. Technical Report. UMI Order Number: TR87-811, Cornell University (1987)

[5] Gupta, I., B., K. and Van Renesse, R.: Fighting fire with fire: Using Randomized Gossip to Combat Stochastic Scalability Limits. J. Quality and Reliability Engineering International. **18** (2002)165–184

[6] Hadzilacos, V., and Toueg, S. : A Modular Approach to Fault-tolerant Broadcasts and Related Problems. Technical Report TR 94-1425, Cornell University, Dept. of Computer Science, Cornell University, Ithaca, NY 14853 (1994)
[7] Kalantar, M. H. and Birman K.: Causally ordered multicast: The conservative approach. Proceedings of the International Conference on Distributed Computing Systems. IEEE Computer Society Press, Los Alamitos, CA (1999)
[8] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM. **21** (1978) 558–565
[9] Muñoz, I., Arévalo, S.: Resolución del efecto convoy en arquitecturas jerárquicas de comunicación causal. JCSD (2010)
[10] Raynal, M., Schiper, A., and Toueg, S.: The causal ordering abstraction and a simple way to implement it. J. Inf. Process. Lett. **39** 6 (1991) 343-350
[11] Schiper, A., Birman, K., and Stephenson, P.: Lightweight Causal and Atomic Group Multicast. ACM Trans. Comput. Syst.,**9** 3 (1991) 272 – 31